Productionizing R with Plumber

Prepared for the NSF

Nov 27th, 2019

Josiah Parry

RStudio

You've made wonderful tools in R.

Now what?

RStudio®

Productionization

- Example: we've produced series of R models
- Question: how do we productionize this?
- But what *is* productionization
  - Probably a buzz-word

“Code that isn't going to make the operations team cry”

“It runs in some useful way (all defined based on context) without you babysitting it.”

“ To me it both denotes a streamlined and autonomous running of said model, but also having taken care of a lot things around the model, such as model versioning, retraining and rollback procedure, access logging etc.”

Productionization

1. Best practices
    a. Auditing, versioning, reproducibility, risk-mitigation
2. Making your tools available to others
    a. I.e. analysis, developers, or other machines

Solution? RESTful APIs

AP-Who?

APIs

- Application programming interface
- Aka “machines talking to other machines”
- “Most simply, API’s are ways to — from your computer — call a function on another computer.” - Heather Nolis

APIS

- APIs are a language agnostic way of interacting with tools
- RESTful APIs allow you to write browser queries (http)
- We can:
    - Retrieve information
    - Send information
    - Ask computers to do things

Let’s make it real (ish)

RStudio

Example

- You must identify if Old Town Road is or isn't country
- Tasks
  - Assess the lyrics
  - Assess the audio
  - Create an ensemble model
- Key packages: **plumber** & **pins**

Analysis Plan

- Identify popular Country and Rap songs in the past 3 years {bbcharts}
- Create classifiers based on:
  - Lyrics
  - Audio
  - ensemble

Lyric based model

- Retrieve lyrics {genius}
- Create topic model {topicmodels}
- Use topic model predictions features {tidymodels}
- Train decision tree {tidymodels}

Audio Based Model

- Retrieve audio features {spotifyr}
- Pre-process model {recipes}
- Train decision tree {tidymodels}

Ensemble Model

- “Stacked approach”
  - The predictions from each model are the inputs
- Calculate predictions from each model
- Train decision tree

Productionizing - Step 1

make it functional

Make it functional

- Identify important component
- What will be repeated?
- Make it a function

Productionizing - Step 2

identify and store important objects

RStudio®

Storing Objects

- Certain objects take a while to reproduce
- Think of model objects
- Pre-processed data

Pins

RStudio®

Pins

- Store R objects or other resources
- Local
- Remote
  - GitHub
  - Kaggle
  - RStudio Connect
- Think of things you'll want to call frequently-ish

RStudio®

Productionizing - Step 3

make it an API

Anatomy of a Plumber API

- {roxygen}-like commenting: `#*`

```
library(plumber)

#* @apiTitle Your title goes here
#* @apiDescription Your description goes here
#* @apiVersion 0.1.0
```

Anatomy of a Plumber API

```
#* API endpoint description
#* @param Description of parameter
#* @request-type /api-endpoint

function(param) {
    # code that does stuff
    # this is what is executed at .../api-endpoint
}

r <- plumb()
r$run(port = 8000) # or whatever port you'd like
```

RStudio

Why we make things functional

- Have a number of function already with parameters
- Make these parameters available from the API
- Plug and play :)
- Be concerned with package management, not API management
- I’d recommend debugging R code than R code deployed as an API

SO….

Is it country or not?

Extending to Python

RStudio